f0 Language

AST · Calculator · Parser · Runner

1. AST

Three expression types: a number literal, addition of two expressions, and a conditional (if0) with three sub-expressions. Zero is the only truthy value.

```
(define-type Exp
  [num  (n     : Number)]
  [plus (left  : Exp) (right : Exp)]
  [cnd  (test  : Exp) (then  : Exp) (else : Exp)])
```

2. Calculator (Exp -> Number)

Pattern-matches on the AST. For cnd, only one branch is ever evaluated: the then-branch if the test is zero, the else-branch otherwise.

```
(define (calc e)
  (type-case Exp e
    [(num n)     n]
    [(plus l r)  (+ (calc l) (calc r))]
    [(cnd c t e) (if (zero? (calc c))   ; 0 is TRUE
                     (calc t)
                     (calc e))]))
```

3. Parser (S-Exp -> Exp)

Converts surface syntax into an Exp. Concrete syntax: numbers, {+ e e}, {if0 e e e}.

```
(define (parse s)
  (cond
    [(s-exp-number? s)
     (num (s-exp->number s))]
    [(s-exp-list? s)
     (let ([l (s-exp->list s)])
       (cond
         [(symbol=? '+ (s-exp->symbol (first l)))
          (plus (parse (second l))
                (parse (third  l)))]
         [(symbol=? 'if0 (s-exp->symbol (first l)))
          (cnd  (parse (second l))
                (parse (third  l))
                (parse (fourth l)))]
         [else
          (error 'parse "unknown operator")]))]
    [else
     (error 'parse "not a number or list")]))
```

4. Runner (S-Exp -> Number)

Composes parse and calc into a single pipeline entry point.

```
(define (run s)
  (calc (parse s)))
```

5. Tests

```
(print-only-errors #true)

; --- num and plus ---
(test (run `1)              1)
(test (run `2.3)            2.3)
(test (run `{+ 1 2})        3)
(test (run `{+ {+ 1 2} 3})  6)

; --- if0: zero is TRUE → take then-branch ---
(test (run `{if0 0 1 2})           1)
(test (run `{if0 {+ 0 0} 99 42})   99)

; --- if0: non-zero is FALSE → take else-branch ---
(test (run `{if0 1 1 2})           2)
(test (run `{if0 {+ 1 2} 10 20})   20)

; --- nested if0 ---
(test (run `{if0 {if0 0 0 1} 100 200})  100)
(test (run `{if0 {if0 1 0 5} 100 200})  200)

; --- parse errors ---
(test/exn (parse `{* 1 2}) "unknown operator")
```